INFO-P016

Patent

UNITED STATES PATENT APPLICATION

for

METHOD AND SYSTEM FOR PROVIDING TRANSFER OF ANALYTIC APPLICATION DATA OVER A NETWORK

Inventors:

MOHAN SANKARAN
VOLODYMYR BUTSKY
SRIDHAR C. KORITALA
ZHENYU TANG

prepared by:

WAGNER, MURABITO & HAO LLP
Two North Market Street
Third Floor
San Jose, CA 95113
(408) 938-9060

METHOD AND SYSTEM FOR PROVIDING TRANSFER OF ANALYTIC APPLICATION DATA OVER A NETWORK

FIELD OF THE INVENTION

The invention relates generally to computer system networks, and more particularly, to securing rapid, reliable, and private communication between networked computers in a multiple data warehouse/analytic application environment.

BACKGROUND OF THE INVENTION

Computers are used to perform a wide variety of applications in such diverse fields as finance, traditional and electronic commercial transactions, manufacturing, health care, telecommunications, etc. Most of these applications typically involve inputting or electronically receiving data, processing the data according to a computer program, then storing the results in a database, and perhaps transmitting the data to another application, messaging system, or client in a computer network. As computers become more powerful, faster, and more versatile, the amount of data that can be processed also increases.

Unfortunately, the raw data found in operational databases often exist as rows and columns of numbers and codes which, when viewed by individuals, appears bewildering and incomprehensible. Furthermore, the scope and vastness of the raw data stored in modern databases is overwhelming to a casual observer. Hence, applications were developed in an effort to help interpret, analyze, and compile the data so that it may be readily and easily understood by a human. This is accomplished by sifting, sorting, and summarizing the raw data before it is presented for display, storage, or transmission. Thereby, individuals can now interpret the data and make key decisions based thereon.

Extracting raw data from one or more operational databases and transforming it into useful information (e.g., data "warehouses" and data "marts") is the function of analytic applications. In data warehouses and data marts, the data are structured to satisfy decision support roles rather than operational needs. A data warehouse utilizes a business model to combine and process operational data and make it available in a consistent way. Before the data are loaded into the data warehouse, the corresponding source data from an operational database are filtered to remove extraneous and erroneous records; cryptic and conflicting codes are resolved; raw data are translated into something more meaningful; and summary data that are useful for decision support, trend analysis and modeling or other end-user needs are pre-calculated. A data mart is similar to a data warehouse, except that it contains a subset of corporate data for a single aspect of business, such as finance, sales, inventory, or human resources.

In the end, the data warehouse or data mart is comprised of an "analytical" database containing extremely large amounts of data useful for direct decision support or for use in analytic applications capable of sophisticated statistical and logical analysis of the transformed operational raw data. With data warehouses and data marts, useful information is retained at the disposal of the decision makers and users of analytic applications and may be distributed to data warehouse servers in a networked system. Additionally, decision maker clients can retrieve analytical data resident on a remote data warehouse servers over a computer system network.

An example of the type of company that would use data warehousing is an online Internet bookseller having millions of customers located worldwide whose book preferences and purchases are tracked. By processing and warehousing these data, top executives of the bookseller can access the processed data from the data warehouse, which can be used for sophisticated analysis and to make key decisions on how to better serve the preferences of their customers throughout the world.

The rapid increase in the use of networking systems, including Wide Area Networks (WAN), the Worldwide Web and the Internet, provides the capability to transmit operational data into database applications and to share data contained in databases resident in disparate networked servers. For example, vast amounts of current transactional data are continuously generated by business-to-consumer and business-to-business electronic commerce conducted over the Internet. These transactional data are routinely captured and collected in an operational database for storage, processing, and distribution to databases in networked servers.

The expanding use of "messaging systems" and the like enhances the capacity of networks to transmit data and to provide interoperability between disparate database systems. Messaging systems are computer systems that allow logical elements of diverse applications to seamlessly link with one another. Messaging systems also provide for the delivery of data across a broad range of hardware and software platforms, and allow applications to interoperate across network links despite differences in underlying communications protocols, system architectures, operating systems, and database services. Messaging systems and the recent development of Internet access through wireless devices such as enabled cellular phones, two-way pagers, and hand-held personal computers, serve to augment the transmission and storage of data and the interoperability of disparate database systems.

In the current data warehouse/data mart networking environment, one general concern involves the sheer volume of data that must be dealt with. Often massive, multi-terabyte data files are stored in various server sites of data warehouses or in operational databases. Transmitting these massive amounts of data over WANs or the Internet is a troublesome task. The time needed to move the data is significant, and the probability that the data may contain an error introduced during transmission is increased. Also, the data are also vulnerable to interception by an unauthorized party. Furthermore, when the connection is lost in the process of transmitting the data over a network, there often is a need to retransmit large amounts of data already transmitted prior to the loss of connection, further increasing the time needed to move the data.

Accordingly, there is a need for a reliable, secure, authenticated, verifiable, and rapid system and/or method for the transmission of huge amounts of data, such as data in a data warehouse/mart, over networks such as WANs and the Internet. The present invention provides a novel solution to this need.

SUMMARY OF THE INVENTION

The present invention satisfies a currently unmet need in a networked data warehouse/analytic application environment to provide a method and system that provide reliable, secure, authenticated, verifiable, and rapid system and method for the transmission of huge amounts of data over a network (e.g., operational data, and transformed data in a data warehouse/data mart). The data can be moved from a source to a target (e.g., from a server to a client, or from a client to a server) in the computer system network. The source represents any centralized source on the network, while the target can represent a remotely located device (e.g., at a customer site) or a local device (e.g., a device in communication with the source via a local area network).

In one embodiment, in a source (e.g., server) computer system, an incoming request is received from a target (e.g., client) for a large amount of data (e.g., a data file) resident in a mass storage unit on the server, for example. The incoming request is authenticated and is then used to spawn a session thread between the server and the client. The incoming request includes a command that, in one embodiment, uses Extensible Markup Language (XML). The command is parsed and translated into a set of tasks which can be executed by the server as part of the session thread.

In one embodiment, the data are separated into blocks, and each block is sequentially compressed and encrypted, then sent to the client. In one embodiment, the blocks are processed in parallel, saving time. The transfer of the data to the client is checked to make sure it is complete and accurate.

On the client side, the session thread between the server and client is spawned in response to a message from the server. The compressed and encrypted data blocks are received from the server, then decompressed, decrypted, and assembled into the requested data.

The present invention provides a recovery mechanism for automatically or manually restoring a connection between the server and client when the connection is lost. As part of the recovery mechanism, data transfer is resumed from the point where it was terminated when the connection was lost, so that previously transmitted data do not have to be retransmitted.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

Figure 1A illustrates a schematic block diagram of an exemplary client/server computer system network upon which embodiments of the present invention may be implemented.

Figure 1B illustrates an exemplary computer system upon which embodiments of the present invention may be practiced.

Figure 2A illustrates a general functional block diagram of a computer system network in accordance with one embodiment of the present invention.

Figure 2B illustrates a more detailed functional block diagram of the computer system network generally illustrated in Figure 2A.

Figure 3A illustrates data flow through a first embodiment of an output channel of the present invention.

Figure 3B illustrates data flow through a first embodiment of an input channel of the present invention.

Figure 4A illustrates data flow through a second embodiment of an output channel of the present invention.

Figure 4B illustrates data flow through a second embodiment of an input channel of the present invention.

Figure 5 illustrates data flow through one embodiment of a session thread in accordance with the present invention.

Figures 6A, 6B, and 6C illustrate data transfer recovery after a failure of a network connection in accordance with one embodiment of the present invention.

Figure 7A is flowchart of the steps in a server-side process for transferring data over a network in accordance with one embodiment of the present invention.

Figure 7B is flowchart of the steps in a client-side process for transferring data over a network in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION

Reference will now be made in detail to the preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings. While the invention will be described in conjunction with the preferred embodiments, it will be understood that they are not intended to limit the invention to these embodiments. On the contrary, the invention is intended to cover alternatives, modifications and equivalents, which may be included within the spirit and scope of the invention as defined by the appended claims. Furthermore, in the following detailed description of the present invention, numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be obvious to one of ordinary skill in the art that the present invention may be practiced without these specific details. In other instances, well-known methods, procedures, components, and circuits have not been described in detail so as not to unnecessarily obscure aspects of the present invention.

Some portions of the detailed descriptions that follow are presented in terms of procedures, logic blocks, processing, and other symbolic representations of operations on data bits within a computer memory. These descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. In the present application, a procedure, logic block, process, or the like, is conceived to be a self-consistent sequence of steps or instructions leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, although not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated in a computer system. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as sessions, objects, blocks, parts, threads, or the like.

It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussions, it is appreciated that throughout the present invention, discussions utilizing terms such as "establishing," "issuing," "authenticating," "spawning," "transmitting," “accumulating,” “restoring,” resuming,” “translating,” “storing,” “executing,” “receiving,” “writing,” “compressing,” “decompressing,” “encrypting,” “decrypting,” “sending,” “verifying,” or the like, refer to actions and processes (e.g., processes 700 and 750 of Figures 7A and 7B, respectively) of a computer system or similar electronic computing device. The computer system or similar electronic computing device manipulates and transforms data represented as physical (electronic) quantities within the computer system memories, registers or other such information storage, transmission or display devices. The present invention is well suited to the use of other computer systems.

Figure 1A illustrates a block diagram of client/server computer system network 100 upon which embodiments of the present invention may be practiced. This server/client system 100 is made up of server computer system 110 (e.g., Unix or NT server computer), client computer system 102, and remote computer systems 103-105, (e.g., personal computers, laptop computers, workstations, terminals, etc.) which may be used to access the information accessible to server computer system 110. Server 110 can represent any centralized source on the network 100, while client 102 and remote computer systems 103-105 can represent a remotely located device (e.g., at a customer site) or a local device (e.g., a device in communication with server 110 via a local area network).

Each remote computer system 103-105 has its own physical memory system (e.g., hard drive, random access memory, read only memory, etc.) for storing and manipulating data. Client computer system 102, server computer system 110, and remote computer systems 103-105 are connected for intercommunication and transfer of data by network bus 107. However, it is appreciated that these devices may instead by coupled in a wireless network.

Server computer system 110 is coupled to server mass storage device 112 that is or is not accessible by client computer system 102 and computer terminals 103-105 through network bus 107 directly. Client system 102 also has its own client mass storage device 170. The present invention includes threads and objects within the application software that are executed by server computer system 110 and/or client system 102 to transfer data therebetween (refer to Figure 2B, below).

Located within mass storage device 112 is operational database 116a, which receives and stores the current raw data for a data mart or data warehouse. Raw data received and stored within operational database 116a are transformed by an analytic application into information that is more meaningful for decision support. Data marts/warehouses 113a, located within mass storage device 112, include transformed data processed by the analytic application. It is important to point out that data marts/warehouses 113a and operational database 116a could each reside within a separate mass storage devices and each mass storage device could be connected by network bus 107 to a separate server.

A data file 120 is a file stored within either operational database 116a, within the database of data warehouse/data mart 113b, or elsewhere in server mass storage device 112. In accordance with the present invention, data file 120 is securely, quickly and reliably transmitted over network bus 107 to client computer system 102 or to remote computer systems 103-105 for display or storage on these systems or for use in analytic applications resident on these systems. Data file 120 is a large file containing, for example, operational data such as customer data or third party data. Data file 120 may instead contain data transformed according to an analytic application. It is appreciated that the present invention also can be used to transmit a data stream from server computer system 110 to a target device (e.g., client computer system 102 or to remote computer systems 103-105).

Operational database 116b and data warehouse 113b are also shown residing within client mass storage device 170. A data file 120 is shown also residing in client mass storage device 170 to represent the transmission and receipt of the data file as mentioned in the preceding paragraph. It is appreciated that the present invention can likewise be used to transmit a data file (or a data stream) from client 102 to server 110, or from these devices to any other device on network 100. Generally speaking, the present invention can be used to transmit a data file or a data stream from a source device to a target device. For simplicity of discussion, the present invention is described in the context of a transfer of a data file from a server to a client.

Refer now to Figure 1B, which illustrates an exemplary computer system 1090 upon which embodiments of the present invention may be practiced.

Computer system 1090 exemplifies server 110, client 102, and remote computer systems 103-105 of Figure 1A.

In general, computer system 1090 of Figure 1B comprises bus 1000 for communicating information, one or more processors 1001 coupled with bus 1000 for processing information and instructions, random access (volatile) memory (RAM) 1002 coupled with bus 1000 for storing information and instructions for processor 1001, read-only (non-volatile) memory (ROM) 1003 coupled with bus 1000 for storing static information and instructions for processor 1001, data storage device 1004 such as a magnetic or optical disk and disk drive coupled with bus 1000 for storing information and instructions, an optional user output device such as display device 1005 coupled to bus 1000 for displaying information to the computer user, an optional user input device such as alphanumeric input device 1006 including alphanumeric and function keys coupled to bus 1000 for communicating information and command selections to processor 1001, and an optional user input device such as cursor control device 1007 coupled to bus 1000 for communicating user input information and command selections to processor 1001. Furthermore, an optional input/output (I/O) device 1008 is used to couple computer system 1090 onto, for example, a network.

Display device 1005 utilized with computer system 1090 may be a liquid crystal device, cathode ray tube, or other display device suitable for creating graphic images and alphanumeric characters recognizable to the user. Cursor control device 1007 allows the computer user to dynamically signal the two-dimensional movement of a visible symbol (pointer) on a display screen of display device 1005. Many implementations of the cursor control device are known in the art including a trackball, mouse, joystick or special keys on alphanumeric input device 1006 capable of signaling movement of a given direction or manner of displacement. It is to be appreciated that the cursor control 1007 also may be directed and/or activated via input from the keyboard using special keys and key sequence commands. Alternatively, the cursor may be directed and/or activated via input from a number of specially adapted cursor directing devices.

Figure 2A illustrates a functional block diagram of an embodiment of the client/server computer system network 100 of Figure 1A in accordance with one embodiment of the present invention. In this embodiment, with reference also to Figure 1A, the present invention provides a secure data stream 118 that transfers data file 120 from server computer system 110 to client computer system 102. Data file 120 is originally resident in data warehouse 113a or operational database 116a in server mass storage device 112, and is transmitted to client mass storage device 170.

For simplicity of discussion, communication is shown as occurring from server 110 to client 102; however, it is appreciated that communication can similarly occur from client 102 to server 110. In this latter case, the arrows indicating the direction of data flow would be in the opposite direction, the "output channel" would refer to the channel on client 102, and the "input channel" would refer to the channel on server 110.

Figure 2B is a functional block diagram providing additional details of the client/server computer system network 100 in accordance with one embodiment of the present invention. Listener object 130a is designed to receive an incoming connection request and to spawn a session thread 140a in response (specifically, listener object 130a calls session manager object 138a to create session thread 140a). Server listener object 130a receives, at a well known port 132, an incoming request 134 from client computer system 102. In the present embodiment, the request 134 is generated in the session thread 140b executing on client 102. The request 134 is a request to establish a client socket connection 136 and to transmit data from server mass storage device 112 (e.g., data file 120) to client computer system 102.

A request 134 may be received from a remote device on the network 100, or from a local device. That is, for example, a request 134 may be received over the Internet from a device that is outside of a firewall or a company's intranet (e.g., a local area network), or the request 134 may be from a local device within the company's intranet. However, in the present embodiment, instead of trying to determine the source of request 134, all requests are delivered/received in encrypted form.

In the present embodiment, all computers in the network 100 are considered as non-secure. All keys are stored in encrypted form, with a password-based encryption algorithm used to encrypt and decrypt keys.

All registered users seeking to implement the data retrieval process of the present invention have their own account. Keys for each account are stored in separate files. Each file has the name of the account and is encrypted using the account password. All registered account keys are stored in a central repository (not shown) and are encrypted using a repository password. In the present embodiment, the repository is initialized with a single password hidden and secured by an administrator password. The repository password is used for all secured (encrypted) objects.

The central repository resides in main memory of the server 110 (and likewise, a central repository resides in main memory of client 102). The central repository has the capability to encrypt any stored object using password-based encryption. In one embodiment, the central repository asks the object if it needs encryption.

In the present embodiment, there are three repository object types: an object to represent an account, an object that represents a repository password for the administrative account, and an object to represent each session instance for recovery purposes. The object representing an account contains a key (or password) that is used to establish a secure connection between client 102 and server 110 of Figures 2A and 2B. The object representing the repository password for the administrative account is used to gain access to all repository objects, and is secured with an administrator account password. The object representing each session instance contains all needed information for session recovery (refer to Figures 6A, 6B and 6C, below).

With reference again to Figure 2B, in the present embodiment, listener object 130a is part of a multi-thread software application that is capable of parallel execution in coordination with other threads of the present invention. A "thread" is a part of the application that can execute independently of the other parts that may be executing simultaneously in this parallel mode. Parallel execution of threads in this embodiment of the present invention is accomplished by sharing as much as possible of the application execution between the different threads. Such a multi-threading embodiment increases the speed of the transfer of data in the present invention.

Listener object 130a establishes client socket connection 136 at well known port 132 requested by client computer system 102 on server computer system 110. When a user request 134 is received, listener thread 130a calls the session manager object 138a to create a new session thread and to start to handle the user request. After completing these tasks, listener object 130a returns to the task of listening for another incoming user request (e.g., another request from client computer system 102 or from another computer system in network 100 of Figure 1A).

In the present embodiment, session manager object 138a of Figure 2B spawns the session thread 140a that reads a command from the connection that is established by listener object 130a, parses the command, and executes it (refer also to Figure 5, below). Session manager object 138a includes (tracks) information regarding all of the ongoing sessions.

A session 140a includes a channel object with multiple threads (e.g., channel 139a), over which the data are actually transferred. Channel object 139a is designed to transfer data in a secure manner. Additional information regarding channel object 139a is provided below in conjunction with Figures 3A, 3B, 4A and 4B.

With reference to Figure 2B, for security purposes, only clients authorized to access data warehouse 113a and operational database 116a are allowed to receive data file 120. A protocol incorporated within the present invention is designed to authenticate that incoming request 134 originated at a client computer system 102 that is authorized to receive data file 120. Authentication protocols could include passwords, intelligent tokens, or other well-known techniques.

In the present embodiment, session manager object 138a provides the application program interfaces (APIs) for monitoring and managing sessions. An object is an application data item that includes instructions for the operations to be performed on it. After listener object 130a receives a user request 134, session manager object 138a receives a call from listener object 130a to create and start a session thread 140a for processing user commands. In this embodiment, session manager object 138a spawns session thread 140a.

Session manager object 138a provides the functionality to generate a unique identifier (ID) for a session, and to maintain logs of sessions for recovery purposes. The APIs provided by session manager object 138a include an API to create a session, an API to run the session, an API to stop the session by passing the session ID, an API to update the session's status by passing the session ID, an API to query session information, and an API to recover a failed session.

In the present embodiment, after session manager object 138a creates a new session, it assigns a unique ID to that session. Session manager object 138a saves (e.g., into an internal hash table) a session information object.

Session manager object 138a, when it creates a new session, passes its own reference as an initialization parameter to the session. The session then uses this reference, in combination with its unique session ID, to invoke a session manager callback function to provide an update of its status information to session manager object 138a.

Thus, in the present embodiment of the present invention, listener object 130a of server 110 receives an incoming connection request from client 102 and passes the incoming connection request to the session manager object 138a. Session manager object 138a of server 110 spawns session thread 140a that reads the command from the connection request 134 (e.g., a command requesting transfer of data file 120 of Figure 2A). The command contains all of the information needed to open data file 120 and to send data file 120 through a connection between client 102 and server 110. The session 140a parses the command and creates and initializes a channel object 139a (with its threads), and runs the channel. Channel object 139a will be initialized with the currently established network connection between server 110 and client 102.

Channel object 139a represents the set of objects needed for sending and receiving a file (e.g., data file 120). In the present embodiment, these set of objects include the reader, compressor, encryptor, decompressor, decryptor, and writer objects described in conjunction with Figures 3A-3B and 4A-4B.

On the client side, listener object 139b receives the incoming connection from server 110 and passes this connection to session manager 138b. Session manager 138b spawns a session 140b. Session 140b reads the command from the connection. This command contains all of the information needed to connect to server 110, read the data file 120, and to have the data file 120 sent to client 102. Session 140b parses the command and executes the following: establishes a connection with server 110, sends the command to start the transfer of data file 120, and initializes and runs the channel.

Figure 3A illustrates data flow through an embodiment of an output channel object 139a in accordance with the present invention. In this embodiment, output channel 139a comprises four data transformation threads or objects: reader channel object 142a, compressor channel object 146, encryptor channel object 156, and writer channel object 152a. The data transformers (reader channel object 142a, compressor channel object 146, encryptor channel object 156, and writer channel object 152a can work in parallel (e.g., they each can have their own threads). Output channel 139a also comprises block manager object 154a.

Block manager object 154a contains multiple data blocks (e.g., vectors of data blocks). A data block is designed to store byte arrays between transformations. In the present embodiment, each data block is associated with one data transformation object, and only that object can write to the data block; however, a data transformation object may be associated with multiple data blocks. A size of a data block can vary, so that if a data transformation object finds that it cannot fit output data into the data block buffer, the data transformation object can create a new (larger) buffer that replaces the old (smaller) buffer. Also, for example, a data block containing compressed data can be smaller than one containing uncompressed data.

Block manager object 154a controls the total count of data blocks for each data transformation object; that is, the block manager object 154a has a parameter limiting the number of data blocks per data transformation object. Generally, about four data blocks are specified per data transformation object. If a data transformation object requests a data block, but the amount of available data blocks is equal to the maximum allowed for the transformation object (that is, there are no free data blocks), then block manager object 154a and the data transformation object will wait for a data block to be freed.

Referring still to Figure 3A, in the present embodiment, reader channel object 142a reads a part of data file 120 and writes that part into a first data block buffer in the main memory of server computer system 110. Data file 120 is typically a large file. By reading a only a part of the file, downstream transformations relating to compression and encryption may commence in parallel, while subsequent parts of data file 120 are read and written to first data block buffer.

Compressor channel object 146 reads the data in the first data block buffer, transforms it (compresses it), and writes the compressed data to a second data block buffer. Compressor channel object 146 encodes the data contained in data file 120 in a way that makes it more compact.

Encryptor channel object 156 reads the compressed data from the second data block buffer, encrypts it, and writes it a third data block buffer.

Writer channel object 152a reads the encrypted data block and writes it to the network socket stream (to the input channel 139b; Figure 3B).

In the present embodiment, output channel 139a functions as follows. Output channel 139a receives data file 120. Data file 120 may be received by output channel 139a in its entirety and then broken into smaller blocks of data, or data file 120 may be read from mass storage device 112 (Figure 1A) a portion at a time.

Reader channel object 142a request a free data block. Block manager object 154a searches for the free data block, and if there is no such block, then block manager object 154a creates a new block, marks it as free, and assigns it to reader channel object 142a. Reader channel object 142a writes data from data file 120 to the data block and marks the data as ready for compression. Compressor channel object 146 receives this data block from block manager object 154a and requests a free block (for its output). Block manager object 154a creates a new data block, marks it as free, and assigns it to compressor channel object 146.

In parallel, reader channel object 142a requests another data block, and as described above, block manager 154a creates a new block, marks it as free, and assigns the new data block to reader channel object 142a. Reader channel object 142a can then write another portion of data file 120 to this block and mark it as ready for compression.

In the meantime, compressor channel object 146 compresses (encodes) the data contained in its respective data block, marks it as ready for encryption, and frees its respective data block. Encryptor channel object 156 receives this (compressed) data block from block manager object 154a and requests a free block for its output. Block manager object 154a creates a new data block, marks it as free, and assigns it to encryptor channel object 156.

Encryptor channel object 156 encrypts the data contained in its respective data block, marks it as ready for ready for writing, and frees its respective data block. Writer channel object 152a receives the encoded (compressed) and encrypted data block from block manager object 154a and writes to the network socket output stream 307.

The process described above is repeated until the reader channel object 142a reads the last block of data in data file 120. Each block of data is stepped through output channel 139a, with data blocks being created and freed as needed by the respective transformation objects. In this manner, the number of data blocks can be reduced so that memory resources are not unduly consumed.

In this embodiment of the present invention, means well known to those of ordinary skill in the art are utilized to verify that data file 120 was completely and accurately transmitted to client 102.

Thus, the goals of the present invention are achieved. A data file 120 is sent on request from server computer system 110 to at least one computer system remote from server computer system 110 (e.g., client 102). The data transfer is accomplish securely, rapidly, and reliably.

Figure 3B illustrates data flow through an embodiment of an input channel 139b of the present invention. As shown in Figures 2B and 3B, in another aspect of the present invention, the operations of server computer system 110 may be mirrored on the client computer system 102. A network stream of compressed/encrypted data blocks 307 are received from server computer system 110 by client computer system 102 and are decrypted, decompressed, and ultimately assembled into data file 120. Alternatively, the data blocks may received by client 102 en masse from server 110.

On the client side, reader channel object 142b reads formatted (encrypted and compressed) data from network input stream 307. A decryptor channel object 164 reads data from a data block in block manager object 154b, decrypts the data, and writes the data to a data block in block manager object 154b.

A decompressor channel object 158 reads data from a data block in block manager object 154b, decompresses (decodes) the data, and writes the data to a data block in block manager object 154b. Writer channel object 152 writes the data to the data file 120 output stream to mass storage device 170 (Figure 2A), for example.

The data transformers (reader channel object 142b, decryptor channel object 164, decompressor channel object 168, and writer channel object 152) function similar to that described above in conjunction with Figure 3A. Means well known to those skilled in the art can be utilized to verify that data file 120 was completely and accurately transmitted.

Thus, the goals of the present invention are again achieved in the client-side embodiment of the present invention. A data file 120 is sent on request of client computer system 102 from server computer system 110 to client computer system 102. The data transfer is accomplished securely, rapidly and reliably.

The forgoing discussion illustrates a "pull" of data from the server computer system 110 by client server system 102. As can be appreciated, in another aspect of the present invention, data transfer could be accomplished by "push" of data from the server computer system 110 wherein, server computer system 110 would command a "listening" client computer system 102 to receive data. Appropriate socket connections, threads, and objects would be created in accordance with the present invention, and data would be transfer over the computer network from the server computer system 110 to the client computer system 102.

Figures 4A and 4B illustrate alternative embodiments of output channel 139a and input channel 139b of Figures 3A and 3B (the alternative embodiments are designated output channel 122 and input channel 124). The operation and function of numbered elements in Figures 4A and 4B is the same as like numbered element in Figures 3A and 3B, respectively.

In the embodiment of Figure 4A, a streaming mechanism is used for compressing data file 120, encrypting data file 120, and then sending data file

120 over the network to the client 102 or another remote device. Thus, instead of dividing the file into blocks as described above in conjunction with Figure 3A, and treating each block as a small file, the file can instead be treated as a contiguous file.

The compressor channel object 164 and the encryptor channel object 156 may produce an output that is different in size from their input. The stage output streams 186 and 188 write data to an output data block buffer until that block is full, or until there is no more data to write. If the current output data block is full, then output streams 186 and 188 indicate that the current block is ready to be read by the next transformation object, and asks block manager object 154a for the next available output data block. By accumulating streaming parts of data file 120 from compressor channel object 146 and encryptor channel object 156, better management of data block buffers by block manager object 154 may be realized.

For example, compressor channel object 146 may be capable of a 10:1 compression ratio. Instead of specifying an output data block buffer size equal to one-tenth the size of the input buffer, better use of the data block buffers may be achieved by accumulating ten parts of compressed data blocks in the stage output stream 186 before writing to the output data block buffer, so that the output buffer is sized the same as the input buffer size.

In Figure 4B, similar to the discussion above, stage output stream 190 and stage output stream 192 accumulate data for decryptor channel object 164 and decompressor channel object 158, respectively. Additionally, stage input stream 194 is provided in input channel 124 to receive the entire stream of compressed data blocks until the end of data file 120 is reached, because decompressor channel object 168 may require a complete compressed data file 120 in order to operate.

Figure 5 illustrates data flow through one embodiment of a session thread 140a in a server computer system 110 (Figure 2B) in accordance with the present invention. It is appreciated that a similar data flow occurs through a session thread 140b in a client computer system 102 (Figure 2B). Session threads 140a and 140b are executed under direction of the session manager objects 138a and 138b, respectively (refer to Figure 2B).

Referring to Figure 5, a session thread 140a is created for each incoming request 134. Requests can include commands such as status commands, commands to send one or more files (e.g., data file 120) to one or more remote devices (e.g., client 102), and commands to receive one or more files from a remote device. In one embodiment, the commands in request 134 use the Extensible Markup Language (XML).

In this embodiment, protocol 174 validates incoming request 134 and directs it to XML command translator 173. Protocol 174 is used to send and receive commands in an encrypted format, and is used for authentication. Protocol 174 is used by channel factory 182 to open a communication socket for a channel (e.g., input channels 139b or 124, or output channels 139a or 122 of Figures 3B, 4B, 3A and 4A, respectively).

Continuing with reference to Figure 5, XML command translator object 173 parses the incoming request 134, generates optimized low-level internal tasks, and inserts the tasks with associated parameters into task table 176. Each parameter is passed to all of the tasks that require it using the "declare" task (mentioned below), so that when a parameter is declared, it is shared at multiple locations and in multiple tasks where it is needed.

Incoming request 134 is translated into low-level incoming tasks because the number of these tasks can be limited, while the number of XML commands in incoming request 134 could be large. However, the XML commands can be translated into low-level internal tasks to facilitate implementation and to make the implementation more extensible. Also, use of low-level internal tasks instead of XML commands facilitates parallel processing and avoids the need for redundant processing steps.

The low-level internal tasks include but are not limited to the following:

Connect: to establish a connection between two devices (e.g., server 110 and client 102), to create and initialize protocol 174, and to pass an initialization command to session manager object 138a;

Create Channel: to create and initialize a channel;

Run Channel: to perform the file transfer;

External Execute: to execute an external command on the local device;

Get Session Status: to get a status report on one or more sessions;

Stop Session: to stop a session;

Declare: to insert a row in a separate variable vector maintained by task executor 178 (the variable vector provides a mechanism to share objects across multiple tasks);

Wait: to wait until a channel has finished transferring a file or files;

Terminate: to terminate a session;

Create Account: to create a new account;

Edit Account: to edit an existing account; and

Remove Account: to remove an existing account.

XML command translator 173 places these tasks into task table 176 for execution by task executor 178. Task table 176 is a memory object and comprises a hash table of the tasks.

Task executor 178 executes the tasks from task table 176 in a given order. Task executor 178 uses an API of session manager object 138a (Figure

2B) to execute the tasks. Task executor 178 executes in a loop through task table 176 until terminate command is found. Once XML command translator 173 creates a task table 176 for a command 141, task executor 178 takes control of the session thread and starts executing the tasks in the session thread. Task executor 178 also updates session statistics for the recovery option described below in conjunction with Figures 6A, 6B, and 6C.

With reference to Figure 5, a channel object (e.g., channel object 139a, and also channel object 122 of Figure 4A) is generated and initialized by a channel factory 182. In the present embodiment, channel factory 182 initializes channel object 139a with input and/or output streams.

Continuing with reference to Figure 5, channel object 139a represents the set of data transformation objects needed for sending and receiving a file (e.g., data file 120). In the present embodiment, the set of data transformation objects includes the reader, compressor, encryptor, decompressor, decryptor, and writer objects described in conjunction with Figures 3A-3B and 4A-4B. Protocol 174 direct executed tasks to a remote session 184.

Figures 6A, 6B and 6C illustrate another aspect of the present invention relating to data transfer recovery after a temporary loss and/or failure of a network connection. Because large amounts of data are being transferred in accordance with the present invention, recovery is an important consideration.

In the present embodiment, two recovery modes are considered: automatic network connection recovery, and manual session recovery.

Automatic network connection recovery means that both the input channel 139b and the output channel 139a (Figure 2B) are running but the network connection is lost or has failed. In this case, the network connection can be recovered automatically.

When a network connection fails, both channels get notification via an "exception." When a channel receives an exception, it calls the API for its respective session manager (e.g., session manager object 138a or 138b of Figure 2B) to restore the connection. The API returns a new session thread for the connection if the connection can be restored, or a NULL if the connection cannot be restored.

Referring to Figures 6A, 6B and 6C, there are two session threads involved with the recovery process. In Figure 6A, session manager object 138a (Figure 2B) creates session 1 (e.g., session thread 140a) on server computer system 110 and initiates the connection with client computer system 102. Session manager object 138b (Figure 2B) receives the request to start a session and creates session 1 (e.g., session thread 140b) on client computer system 102.

In Figure 6B, the connection is lost. In Figure 6C, once the connection is lost and the initiating session manager (session manager object 138a on server 110) gets called to recover the session, session manager object 138a sends a request to client 102 (session manager object 138b) to restore the network connection for session 1. When client 102 (specifically, session manager object 138b) receives the request from server 110, it spawns a second session thread (session 2, e.g., session thread 140c). Session 2 passes the connection to the waiting session 1 on client 102. Once the connection is restored, both channels (output channel 139a and input channel 139b) continue to transfer data from the point at which they stopped.

Information about a session is stored in session manager objects 138a on server 110. Session manager 138a stores all session parameters and the count of bytes of data written in session 1 before the failure. Accordingly, reading can start reading from the correct byte offset, and writing can proceed by appending those data blocks created but not yet sent.

In the recovery mode, task table 176 (Figure 5) contains the same tasks but, when executed, some of the tasks may be skipped depending on the results of previous task execution and settings.

Manual session recovery is used when either of the channels fails. Recovery typically does not occur automatically because it is desirable to first determine and resolve the cause of the failure.

Figure 7A is a flowchart of the server side steps in a process 700 for transferring data over a network 100 (Figure 1A) in accordance with one embodiment of the present invention. In this embodiment, process 700 is implemented by a server computer system 110 (Figure 1A), exemplified by computer system 1090 (Figure 1B), as computer-readable instructions stored in a memory unit (e.g., ROM 1003, RAM 1002 or data storage device 1004 of Figure 1B) and executed by a processor (e.g., processor 1001 of Figure 1B). However, it is appreciated that some aspects of process 700 may be implemented on server computer system 110 with other aspects of the process 700 performed on client computer system 102 (Figure 1A).

In step 702 of Figure 7A, server 110 receives a request 134 (Figure 2B) for a data file residing in a mass storage unit on the server (e.g., data file 120 residing in mass storage device 112 of Figure 1A). In one embodiment, a listener object 130a (Figure 2B) is listening for such a request. In the present embodiment, the request 134 includes commands; refer to Figure 5. In one embodiment, request 134 uses XML.

In step 704 of Figure 7A, the request 134 is authenticated to make sure that the request is from an authorized user. In one embodiment, protocol 174 (Figure 5) validates incoming request 134.

In step 706 of Figure 7A, a session thread (e.g., session thread 140a) is spawned in response to the user request 134 (Figure 2B). In one embodiment, listener object 130a (Figure 2B) calls session manager object 138a (Figure 2B), and session manager object 138a creates session thread 140a and assigns a unique ID to it. Session thread 140a reads the command(s) contained in the user request 134, and translates the request 134 into a set of low-level incoming tasks that are to be executed for session thread 140a (refer to Figure 5). Session manager 138a also sends a message to client 102 directing it to spawn a session thread (e.g., session thread 140b). A channel object 139a is also generated, providing the set of data transformation objects needed for sending and receiving data file 120; refer to Figures 3A and 4A.

In step 708 of Figure 7A, the data in data file 120 are read from server mass storage device 112 (Figure 1A) and compressed as described in conjunction with either Figure 3A or Figure 4A.

In step 710 of Figure 7A, the compressed data are encrypted as described in conjunction with Figure 3A or Figure 4A.

In step 712 of Figure 7A, the data are sent to the requesting device (e.g., to client 102 over network bus 107 in network 100 of Figure 1B), and complete and accurate data transfer are verified.

Steps 708, 710 and 712 are performed in parallel for different parts of the data file 120.

Figure 7B is a flowchart of the client side steps in a process 750 for transferring analytical data over a network in accordance with one embodiment of the present invention. In this embodiment, process 750 is implemented by client computer system 102 (Figure 1A), exemplified by computer system 1090 (Figure 1B), as computer-readable instructions stored in a memory unit (e.g., ROM 1003, RAM 1002 or data storage device 1004 of Figure 1B) and executed by a processor (e.g., processor 1001 of Figure 1B). However, it is appreciated that some aspects of process 750 may be implemented on client computer system 102 with other aspects of the process 750 performed on server computer system 110.

In step 752 of Figure 7B, a requesting device (e.g., client 102 of Figure 1A) issues a request 134 (Figure 2B) to a server (e.g., server 110 of Figure 1A) for a data file 120 (Figure 1A).

In step 754 of Figure 7B, client 102 receives from server 110 (specifically, from session manager object 138a of Figure 2B) a message directing client 102 to spawn a session thread (e.g., session thread 140b of Figure 2B).

In step 756 of Figure 7B, client 102 receives from server 110 encrypted and compressed data blocks that represent data file 120; refer to Figure 3B or Figure 4B.

In step 758 of Figure 7B, the data are decrypted as described by Figure 3B or Figure 4B.

In step 760 of Figure 7B, the data are decompressed as described by Figure 3B or Figure 4B.

Steps 756, 758 and 760 are performed in parallel for different parts of the data file 120.

In summary, the present invention provides a reliable, secure, authenticated, verifiable, and rapid system and method for the transmission of huge amounts of data over a network, such as the data used in an analytic application (e.g., operational data, and transformed data in a data warehouse/data mart).

The foregoing descriptions of specific embodiments of the present invention have been presented for purposes of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed, and obviously many modifications and variations are possible in light of the above teaching. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the Claims appended hereto and their equivalents.